

# MONGODB

# **1. INTRODUCTION**

**QU'EST-CE QUE C'EST ?**



- Serveur de bases de données

- Serveur de bases de données
- NoSQL

- Serveur de bases de données
- NoSQL
- Orienté document

# LES BASES NOSQL



Not Only SQL



- Viennent briser l'hégémonie des SGBDR traditionnels

- Viennent briser l'hégémonie des SGBDR traditionnels
- Très à la mode, mais pas obligatoires !

**POURQUOI LE NOSQL ?**



- Moteurs relationnels pas adaptés aux gros volumes de données

- Moteurs relationnels pas adaptés aux gros volumes de données
- Moteurs relationnels plus difficiles à distribuer



# CARACTÉRISTIQUES

# CARACTÉRISTIQUES

- MongoDB est développé en Open Source par la société [MongoDB, Inc.](#)
- Développé en C++ => bonnes performances
- Gère les documents BSON
- Permet l'indexation secondaire des documents
- Permet le sharding (théoriquement jusqu'à 1000 noeuds)
- Permet la réplication maître-esclave

## **2. INSTALLATION**

# Import de la clef publique

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --re
```

# Création de la liste des fichiers

```
$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubu
```

# Update des paquets locaux

```
$ sudo apt-get update
```

# Installation des paquets MongoDB

```
$ sudo apt-get install -y mongodb-org
```

# Installation d'une version spécifique de MongoDB

```
$ sudo apt-get install -y mongodb-org=4.0.0 mongodb-org-server=4.
```



Toutes les procédures d'installation sur le site de  
l'éditeur :

<https://docs.mongodb.com/manual/installation/>

```
$ mongod --version
```

```
$ sudo systemctl start mongod  
$ sudo systemctl status mongod
```

# **3. SÉCURISATION**

**CRÉER LES UTILISATEURS MONGODB**

# Connexion au shell Mongo

```
$ mongo
```

# Branchement sur la base native admin

```
> use admin
```

# Création du compte administrateur

```
> db.createUser( { user: "admin", pwd: "", roles: [ { role: "re
```

# Branchement sur la base technocite

```
> use technocite
```



# Création du compte dédié pour la base technocite

```
> db.createUser( { user: "technocite", pwd: "", roles: [ { role
```

# Sortie du shell

```
> quit()
```

# Activation de l'authentification

```
$ cd /etc/ && nano mongod.conf
```

```
security:  
  authorization: enabled
```

# Redémarrage du serveur

```
$ sudo systemctl restart mongod
```

# Test

```
$ mongo -u admin -p your_password --authenticationDatabase=admin
```

# **4. UTILISATION**

On attaque le serveur via l'utilitaire *mongo*, fourni par MongoDB





- Via le shell

- Via le shell
- Via un fichier javascript

# Shell

```
$ mongo
```

# Fichier JS

```
$ mongo path-to-js-file.js
```

# **5. PRATIQUE**

**CONNEXION À LA DB**

/demos/05-pratique/01-connexion/index.js

```
db = connect('technocite');  
let dbs = db.adminCommand('listDatabases');  
printjson(dbs);
```

```
$ mongo index.js
```

**DOCUMENT**

**CRÉER UN DOCUMENT**





## EXERCICE

Insérer un étudiant dans la collection *students* de la base de données *technocite*

/exos/05-pratique/02-create-document/index.js

```
db = connect('technocite');

var dbs = db.adminCommand('listDatabases');
print('Liste des DB AVANT insertion du 1er document : \n');
printjson(dbs);

var student = {
  lastname: 'Moissa',
  firstname: 'Marc',
  email: 'marc.moissa@comptoir.be'
}

db.students.insert(student);

var dbs = db.adminCommand('listDatabases');
print('Liste des DB après insertion du 1er document : \n');

$ mongo index.js
```

**RECHERCHER UN DOCUMENT**







# RECHERCHE SIMPLE

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find();  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```



# RECHERCHE AVEC PARAMÈTRE

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find({ firstname : 'marc' });  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# RECHERCHE AVEC PARAMÈTRES

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find({  
  firstname : 'marc',  
  lastname : 'moissi'  
});  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# SYNTAXE ALTERNATIVE

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');
var students = db.students.find(
  $and : [
    { firstname : 'marc' },
    { lastname : 'moissi' }
  ]
);
clients.forEach( (student) => {
  printjson(student)
});
```

```
$ mongo index.js
```

## OPÉRATEURS DE COMPARAISON

Opérateur	Signification
\$gt	<i>Greater Than.</i> Ex { \$gt : 5 }
\$lt	<i>Lower Than.</i> Ex { \$lt : 5 }
\$in	<i>INcluded.</i> Ex { \$in : [0, 1, 2, 3, 4] }
\$gte	<i>Greater Than or Equals.</i> Ex { \$gte : 5 }
\$lte	<i>Lower Than or Equals.</i> Ex { \$lte : 5 }
\$ne	<i>Not Equals.</i> Ex { \$ne : 5 }

# RECHERCHE AVEC OPÉRATEURS DE COMPARAISON

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  { lastname : { $in : [ 'marc', 'valère', 'antoine' ] } }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# SPÉCIFIER UNE CONDITION OR

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  {  
    $or : [ { lastname : 'Lefevre' }, { firstname : 'David' } ]  
  }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# UTILISER OR ET AND CONJOINTEMENT

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');
var students = db.students.find(
  {
    firstname : { $in : [ 'marc', 'valère', 'antoine' ] },
    $or : [ { lastname : 'Van de Velde' } ]
  }
);
clients.forEach( (student) => {
  printjson(student)
});
```

```
$ mongo index.js
```

# RECHERCHER SELON L'EXISTENCE

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  { firstname : { $exists : true } }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```



## RECHERCHER SELON LE TYPE

Type	Valeur
Double	1
String	2
Object	3
Array	4
Binaire	5
Undefined	6 - Deprecated

## RECHERCHER SELON LE TYPE (2E PARTIE)

Type	Valeur
Object id	7
Boolean	8
Date	9
Null	10
Timestamp	17

# RECHERCHER SELON LE TYPE

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  { firstname : { $type : 2 } }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# RECHERCHER AVEC UNE REGEX

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  { $where : "this.firstname.match(/^marc|michel$/i)" }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# RECHERCHER DANS DES SOUS-DOCUMENTS

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  { address.city : 'Paris' }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# RECHERCHER DANS DES TABLEAUX

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  { address.city : 'Paris' }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# TRIER LES DOCUMENTS

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find().sort(  
  { lastname : 1, firstname : 1 }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```

# SPÉCIFIER LA PROJECTION

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  {},  
  { _id : 0, lastname : 1, firstname : 1 }  
);  
clients.forEach( (student) => {  
  printjson(student)  
});
```

```
$ mongo index.js
```



# COMPTER LE NOMBRE DE DOCUMENTS

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var students = db.students.find(  
  {},  
  { _id : 0, lastname : 1, firstname : 1 }  
);  
print students.count(); // Cursor.count()  
print students.length(); // Ne pas confondre avec Array.length
```

```
$ mongo index.js
```

# RECHERCHER UN SEUL DOCUMENT

/demos/05-pratique/03-find-document/index.js

```
db = connect('technocite');  
var student = db.students.findOne(  
  {},  
  { _id : 0, lastname : 1, firstname : 1 }  
);  
printjson(student);
```

```
$ mongo index.js
```

**EDITER UN DOCUMENT**



# EDITER DES DOCUMENTS AVEC UPDATE

/demos/05-pratique/04-update-document/index.js

```
db = connect('technocite');  
var student = {}  
db.students.update(  
  { lastname : 'Moissa' },  
  student  
);  
printjson(student);
```

```
$ mongo index.js
```

# Opérateurs de champs

# Opérateurs de tableaux

/demos/05-pratique/04-update-document/index.js

```
db = connect('technocite');
var student = {}
db.students.update(
  { lastname : 'Moissa' },
  {
    $set : { company : "Triptyk" },
    $inc : { age : 1 }
  },
  { multi : true }
);
printjson(student);
```

```
$ mongo index.js
```



/demos/05-pratique/04-update-document/index.js

```
db = connect('technocite');  
var student = {}  
db.students.update(  
  { lastname : 'Moissa' },  
  { $currentDate : { lastModified : { $type : timestamp } } }  
);  
printjson(student);
```

```
$ mongo index.js
```

**SUPPRIMER UN DOCUMENT**



/demos/05-pratique/05-delete-document/index.js

```
db = connect('technocite');  
var student = {}  
db.students.remove(  
  { lastname : 'Moissa' },  
  { justOne : true }  
);  
printjson(student);
```

```
$ mongo index.js
```

**COLLECTION**



Deux fonctions utiles :

- *mongodump*
- *mongorestore*

**EXERCICE**



Voir exos/02-consignes.txt

# **6. MODULE MONGOOSE**

Interface de modélisation MongoDB pour Node.js

<http://mongoosejs.com/>

# INSTALLATION



**CONNEXION**

```
const mongoose = require('mongoose');
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://username:password@host:port/database'
  if(error) throw error;
  console.log('Mongo is now connected to our system please reques
})
```

**SCHEMA**





- Correspond à une collection

- Correspond à une collection
- Définit la forme des documents de cette collection

- Correspond à une collection
- Définit la forme des documents de cette collection
- Définition clef/valeur

- Correspond à une collection
- Définit la forme des documents de cette collection
- Définition clef/valeur
- La valeur représente le type de donnée

## TYPES DE DONNÉES

- String
- Number
- Buffer
- Boolean
- Mixed
- ObjectId
- Array
- Map

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var clientSchema = new Schema({
  firstname: String,
  lastname: String,
  address: String,
  date: { type: Date, default: Date.now },
  rating: Number,
  hidden: Boolean,
  meta: {
    votes: Number,
    favs: Number
  }
});
const Client = mongoose.model('Client', clientSchema);
```

# QUERIES CRUD



## CRÉER UN DOCUMENT

- Méthode d'instance *save()*
- Méthode de classe *create()*

```
const Client = require('./models/client');  
let client = new Client({ firstname : '', lastname : ''});  
client.save();
```

```
const Client = require('./models/client');  
Client.create({ firstname : '', lastname : ''}, () => {});
```

## CRÉER UN SOUS-DOCUMENT

- L'objet doit faire partie d'une collection du document parent

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const addressSchema = new Schema({
  street : String,
  city : String,
  zip : String
});
const clientSchema = new Schema({
  firstname: String,
  lastname: String,
  address: [addressSchema]
});
const Client = mongoose.model('Client', clientSchema);
```

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const addressSchema = new Schema({
  street : String,
  city : String,
  zip : String
});
const clientSchema = new Schema({
  firstname: String,
  lastname: String,
  address: [
    {
      street : String,
      city : String,
      zip : String
    }
  ]
});
```

**TROUVER UN OU PLUSIEURS DOCUMENTS :**

Méthodes de classe :



Méthodes de classe :

- *find()*

## Méthodes de classe :

- *find()*
- *findByld()*

## Méthodes de classe :

- *find()*
- *findById()*
- *findOne()*

```
const Client = require('./models/client');
let model = Client.find( // Collection
  { age : { $gte : 18 }}, // Query criteria
  { name : 1 , address : 1 } // Projection
)
.limit(5) // Cursor modifier
.sort( { name : -1 } ); // Sorting
```

```
const Client = require('./models/client');
Client.findOne(
  { lastname : 'Roger' },
  { lastname : 1 , fistname : 1 },
  (error, client) => {
    console.log(client);
  }
);
```

```
const Client = require('./models/client');
Client.findById(
  { _id : '1285fs8sf54bgfhyq98fq' },
  { lastname : 1 , fistname : 1 },
  (error, client) => {
    console.log(client);
  }
);
```

**EDITER UN OU PLUSIEURS DOCUMENTS :**

Méthodes de classe :



Méthodes de classe :

- *findOneAndUpdate()*

Méthodes de classe :

- *findOneAndUpdate()*
- *findByAndUpdate()*

Méthodes de classe :

- *findOneAndUpdate()*
- *findByAndUpdate()*
- *updateOne()*

## Méthodes de classe :

- *findOneAndUpdate()*
- *findByAndUpdate()*
- *updateOne()*
- *updateMany()*

## Méthodes de classe :

- *findOneAndUpdate()*
- *findByAndUpdate()*
- *updateOne()*
- *updateMany()*
- *update()*

```
const Client = require('./models/client');
Client.findByIdAndUpdate(
  '5b749af3096bcd4741534c7f',
  { $set : { firstname : 'Toto' } },
  (error, client) => {
    console.log(client); // unaltered document
  }
);
```

```
const Client = require('./models/client');
Client.findByIdAndUpdate(
  '5b749af3096bcd4741534c7f',
  { $set : { firstname : 'Toto' } },
  { new : true },
  (error, client) => {
    console.log(client); // altered document
  }
);
```

```
const Client = require('./models/client');
Client.findOneAndUpdate(
  { lastname : 'Bourne' },
  { $set : { firstname : 'Jason' } },
  { upsert : true, new : true }, // Create new document if not found
  (error, client) => {
    console.log(client); // altered document
  }
);
```



```
const Client = require('./models/client');
Client.updateOne(
  { lastname : 'Nash' },
  { $set : { firstname : 'Arnold' } },
  (error, client) => {
    console.log(client); // Pas de retour de document
  }
);
```

```
const Client = require('./models/client');
Client.update(
  { lastname : 'Nash' },
  { $set : { firstname : 'Arnold' } },
  (error, client) => {
    console.log(client); // Pas de retour de document
  }
);
```

**SUPPRIMER UN OU PLUSIEURS DOCUMENTS :**

Méthodes de classe :

Méthodes de classe :

- *findOneAndDelete()*

Méthodes de classe :

- *findOneAndDelete()*
- *findOneAndRemove()*

## Méthodes de classe :

- *findOneAndDelete()*
- *findOneAndRemove()*
- *findByAndDelete()*

## Méthodes de classe :

- *findOneAndDelete()*
- *findOneAndRemove()*
- *findByAndDelete()*
- *findByAndRemove()*



## Méthodes de classe :

- *findOneAndDelete()*
- *findOneAndRemove()*
- *findByAndDelete()*
- *findByAndRemove()*
- *deleteOne()*

## Méthodes de classe :

- *findOneAndDelete()*
- *findOneAndRemove()*
- *findByAndDelete()*
- *findByAndRemove()*
- *deleteOne()*
- *deleteMany()*

```
const Client = require('./models/client');
Client.update(
  { lastname : 'Nash' },
  { $set : { firstname : 'Arnold' } },
  (error, client) => {
    console.log(client); // Pas de retour de document
  }
);
```